THE UNIVERSITY
OF BIRMINGHAM

School of Computer Science

MSc in Computer Science
2002 - 2003

Rostrum: A Shared Information Space for Wireless and Wired Devices

by
Nickolas Pitsioras
ID 490435
22 September 2003

Contents

CONTENTS 1
ABSTRACT 3
ACKNOWLEDGMENTS 4
INTRODUCTION 5
BACKGROUND MATERIAL 7
An overview of the technologies used 7
ANALYSIS AND SPECIFICATION 9
Requirements of the system 9
What do they mean? 11
From specification to design: what does it look like? 14
OVERALL DESIGN 15
From specification to design: primary design decisions 15
What were the alternatives? 16
What is next? 16
SENDING END: THE J2SE CLIENT 17
What purpose does it serve? 17
How was it designed? 17
Implementing the Client: what does it look like? 18
Understanding the Client functions 18
Components of the Client 20
MICROROSTRUM: THE J2ME CLIENT 22
What purpose does it serve? 22
A necessary overview of the Java 2 Micro Edition (J2ME) 23
How was it designed? 24
Implementing MicroRostrum: what does it look like? 25
Understanding MicroRostrum: how does it function? 26
ROSTRUM: THE DISPLAY SERVER 30
What purpose does it serve? 30
How was it designed? 30
Rostrum: the Network Listeners 30
Accounting and Administration: Program initialization 32
What is a VisualElement? 34
How does it all work together? 35
PROGRAM FILES DEPOT: THE WEB SITE 41
The need for a Web Site 41
What is OTA user initiated provisioning? 41
TESTING ROSTRUM 42
The testing components 42
Clients' Testing 43
Server Testing: the Displayer 44
Network testing 45
PROJECT MANAGEMENT 46
APPRAISAL AND CONCLUSION 47

Overview *47*
Rostrum: the Good, ... *47*
The Bad *48*
And the Ugly *49*
REFERENCES 50
BIBLIOGRAPHY 50
APPENDICES 51
Appendix A: Project Proposal and Timetable *52*
Appendix B: Running the Software *56*
Appendix C: A UML Sequence Diagram of how a Message is posted using Rostrum and J2SE Client *57*

Abstract

Rostrum is a system that acts as a stage for people to express their opinion and interact with each other. It uses a shared medium that can be as small as a computer screen or as large as a wall projection to visualize short messages posted by its users. It uses two versions of a client application and server application. Client applications are intended for either users of small devices such as mobile phones that support the Micro Edition of the Java programming language or users of computers, whether connected to a network via wires or wirelessly, that support the Standard Edition of Java (version 1.4.1).

Keywords: J2ME, J2SE, networking, wireless, mobile phones

Acknowledgments

My gratitude extends to my supervisor Dr. Russell Beale for giving me the freedom to pursue my goals in my own way, while providing me with valuable feedback.

Introduction

Software engineering is a perilous venture. It involves experience, cunning and expertise, much like any other practical endeavor, only it is aimed at enervating one's intellect rather than one's physique.
Such is, therefore, the nature of software engineering that tasks such as scheduling, designing and implementing a system may seem daunting to the eye of the novice programmer; practical experience is the sole remedy for her unease.

Rostrum development offered comparable excitement, as it scurried from design to specification to implementation. The thrust behind the entire system development was the urge to produce a novel application based on newly evolved technologies: wireless networking, mobile phones' applications development.

Rostrum is a system that provides its users a space where they can interact with each other. The interaction is in the form of short text messages that are visualized in a shared medium. This shared medium, the "space" the users share, could be a computer screen, a TV set or a projection on a wall according to what are the requirements of the space the system is ran in.

Users can access their system using any sort of device has capabilities of running a version of the Java runtime environment. These include, but are not limited to, Mobile Phones, Personal Digital Assistants, and Personal Computers as workstations. The supported versions of Java are Java 2 Standard Edition (1.4.1) and Java 2 Micro Edition (1.0.3). All these devices must be connected to or have immediate to the Internet.

The users of the system can "post" their messages in the shared medium where they can be viewed and possibly replied by other users. The shared medium is organized in four virtual sections and to each is assigned a "Topic", where users with similar interests can interact about issues of their common interests. The messages posted by the users are visualized as small round-edged windows (reminiscent of the popular "Post-It" notes).

The system assumes that its users have a minimum exposure to how computers work and how computer applications function. This paper assumes relatively higher level of understanding how computers function, how networks function and possibly a small level of experience with the Java programming language. Every aspect of the system is explained in detail in the following chapters, using screenshots of the system, where necessary. All screenshots were taken under the Windows 2000 Professional operating system.

The structure of this paper is as follows:

The next chapter contains a small overview of the technologies used by Rostrum. Following that is the Analysis and Specification chapter which include the requirements of the system. The Overall Design establishes the need for the division of the system into four components, while the following four chapters introduce, analyze, examine and highlight the various aspects of each component. Finally, the last chapters include the system testing, project management, appraisal and conclusions. These chapters are followed by three appendices, including references and bibliography, the original project proposal and a UML sequence diagram of the normal operation of the system.

Background Material

An overview of the technologies used

Inspecting the trends of recent software development, one may not fail to notice that a large proportion of the systems being evolved are a fine mixture of diverse technologies. Well-accepted and widespread technologies (such as database connectivity, traditional network programming) are often combined with newer, more advanced ones, such as wireless technologies, to create software systems that accomplish the goals of the ever-changing needs of modern software engineering.

The need for this diversity seems to derive from the high specialization and capabilities of modern technologies. The introduction and ever increasing adaptation of wireless computing has played a major role in creating needs of connecting existing devices and systems to a new breed of wireless, network-enabled devices.

Rostrum uses a variety of technologies to realize its functionalities. Being a system based on network connectivity, several aspects of network related technologies are used, applying to both Local Area Networks and the Internet.
No intrinsic knowledge of how each functions is required to operate it. However, a small overview of how *Rostrum* accomplishes connectivity will be given, as a general guide on the principles of the inter-system communication.

● How does *Rostrum* accomplish connectivity?

Rostrum requires a device already connected to a Local Area Network or the Internet. The end user of the Client part of the application for Personal Computers has to configure it by entering the IP address or the hostname of the Display Server. From there on, the communication between Client and Server is taken care of by the Client by using Java Sockets.

> **What is a Socket?**
> ⇒ "A socket is an endpoint for communication between two machines"
> *Java™ 2 Platform Std. Ed. v1.4.1 API*

The Client version for smaller devices, *MicroRostrum*, must use other means for propagating information. This is mainly due to the poor implementation of sockets by the manufacturers of small devices that exist today. The Micro Edition of the Java programming language (J2ME) has basic support for sockets (as opposed to the Standard Edition (J2SE)) as part of its overall philosophy of procuring an equally lightweight version.
Thus, in order to achieve maximum compatibility and untroubled operation, *MicroRostrum* uses the HTTP protocol to communicate with the Display Server. The information sent to the Display Server is embedded in the HTTP header part of the HTTP message that can be safely retrieved and processed.
Every aspect of how the system works will be thoroughly investigated in the following chapters.

- How was *Rostrum* developed?

Rostrum was developed using the Java™ 2 Platform Standard Edition v1.4.1. . The Java programming language is ideal for developing small and medium-sized systems that require networking capabilities. Despite the fact that being part compiled and part interpreted makes it somewhat slower than fully compiled languages (such as the popular C++ language), its native support of multithreading and its powerful network capabilities combined with modern, fast computers, provide an attractive development platform.

- How was *MicroRostrum* developed?

MicroRostrum was developed using the Java™ 2 Micro Edition Platform. The choice of the J2ME platform is beyond rationalization; it's the sole programming platform for the majority of the small devices such as Mobile Phones and Personal Digital Assistants (PDAs). It has remarkable features given that the target devices are very dissimilar and have minor capabilities and processing power. However, the devices themselves are limiting the features any

programming platform can provide. A comprehensive overview of the J2ME platform is included in the following chapters.

> **What is a J2ME?**
> ⇒ "Java 2 Micro Edition or J2ME is a development and runtime environment designed to put Java software on consumer electronics and embedded devices."
> *J. White & D. Hemphill, J2ME, Java in small things, 2002, Manning*

• How was the Web site developed?

The Web site exists for providing some important services that complement *Rostrum.* It provides information about the system, a user guide, a download area where the application files can be downloaded and some screenshots of the different components of the system. The Web site is built using plain static HTML pages and can be accessed and viewed by any regular browser (Microsoft Internet Explorer, Netscape, Mozilla, Opera etc.). The download area serves all versions of the system, though small devices' users will have to manually install the application files. An alternative for these users is to use a user-initiated Over-The-Air (OTA) transaction using a WML link supplied by the web server. This way, the application files can be downloaded and installed automatically to their device. However, the success of this method is subject to Network Operator restrictions (e.g., Orange UK would allegedly not allow this sort of transaction two years ago).

Analysis and Specification

Requirements of the system

The analysis of how the system should be built was based on extensive research into its more obscure areas. On account of having no significant knowledge of

the intrinsic capabilities of technologies such as the Java 2 Micro Edition, the first few weeks of the analysis phase were consumed discovering these.

During the initial period of research, a set of user requirements was formulated, outlining the specifications of the evolved system. By the end of the analysis, this set of requirements had been enriched to accommodate those capabilities of the tools used that would be incorporated in the system. The user requirements definition is as follows:

User and System Requirements

1. To provide a space where short text messages can be displayed and read.
   1.1. To provide a facility where text messages can be individually displayed
   1.2. To ensure that message text is readable for the given display area
2. To provide every device with Java 2 Standard Edition or Java 2 Micro Edition and network capabilities a facility to access the system
   2.1. To provide different versions of the system to match the user's device
   2.2. To provide the same level of services across all devices
3. To enable devices to access the system irrespective of the type of connection
   3.1. To enable wired and wireless devices alike to access the system
4. To provide a facility to enable users access to the application files they need
   4.1. To enable access to all types of devices
5. To make every facility used to access the system configurable by the end user
   5.1. To provide a minimum set of configuration options
   5.2. To enable users to configure personal details
6. To provide facilities by which the space where short messages are displayed consists of a number of smaller spaces with individual boundaries.
   6.1. To enable users to manually select which space to display their message

7. To enable users of non-Java enabled devices to access the system via WAP and HTTP
8. To provide means of configuring system-wide options via a Graphical User Interface (GUI)

What do they mean?

If we tried to capture the goal of the system in a few words, these would be "to enable users to interact via text messages on a shared medium". This simplification is the base on which the system design should begin. The requirements of the system, whether user, system or non-functional, derived from the main functionality of the proposed system, as stated, together with the capabilities and restrictions of the tools used (J2ME, J2SE). The specification of the system will be fashioned after the user and system requirements, as they are explained below.

1st Requirement

The obvious assumption one could make is that this type of system should have a sending end and a receiving end. The receiving end is where the user interaction takes place, in the form of messages, while behind the sending end is obviously the user herself. A user's message should be separated from other users' messages and the content of the message, the text and any other details deemed appropriate, should be clearly visible. Due to the finite space on which the messages are to be displayed, there is a definite tradeoff between message size and visibility versus total number of messages displayed. A formula needs to be worked out in order to maintain a reasonable ratio between the two.

2nd Requirement

This is one of the core assumptions that need to be made in order to maintain high availability levels of the system. Every device, irrespective of size and native support of some version of the Java platform, should be able to access the system. This inexorably leads to the development of two different versions of the software needed, one for J2ME-enabled devices (Mobile Phones, PDAs etc) and

one for J2SE-enabled devices (Laptops, PCs, etc). However, the services that both versions provide should be of the same level. Given the fact that the Micro edition of Java is only a subset of the Standard edition, the final quality of the end products will most probably differ as well.

3rd Requirement

The medium used by the users' devices accessing the system can either be wireless or wired transmission. This fact is a direct result of the IEEE committee's decision to "…make 802.11 compatible with Ethernet above the data link layer. In particular, it should be possible to send an IP packet over the wireless LAN the same way a wired computer sent an IP packet over Ethernet."(Computer Networks, A. S. Tanenbaum, 4th International Edition, 2003, p.69). Thus, neither the user nor the programmer should be able to differentiate her reaction to different network protocols, as these are taken care of by the Java Virtual Machine and the host operating system.

4th Requirement

The nature of the system is such that it should allow users with different types of devices to be able to access the application files needed to access the system at any time. Therefore, a facility must exist that gives users the ability to download the files (since a network connection is a core prerequisite) at any time. The duality of the application files (J2ME and J2SE) must be enforced here as well, in the form of two separate download processes.

5th Requirement

The mobility of the system, on both sending and receiving sides, dictates the existence of a set of configuration options. The receiving side shouldn't need location-specific configuration, rather than runtime configuration, as the location options are delegated to the sending side. Thus, the location specifics are manually configured by the user and should form the absolute minimum requirement. Other details, provided by the user, should be optional and assume default values if not explicitly changed.

6th Requirement

Since the system can form a hub for interaction, care should be taken to divide this into groups of similar interests. Therefore, the visible area of the shared medium can be partitioned accordingly and give the participating users the ability to choose where they would want their short message to be placed. This division can enhance the idea of "conversation" and give users the option to really experience the environment that the system wants to create.

7th Requirement

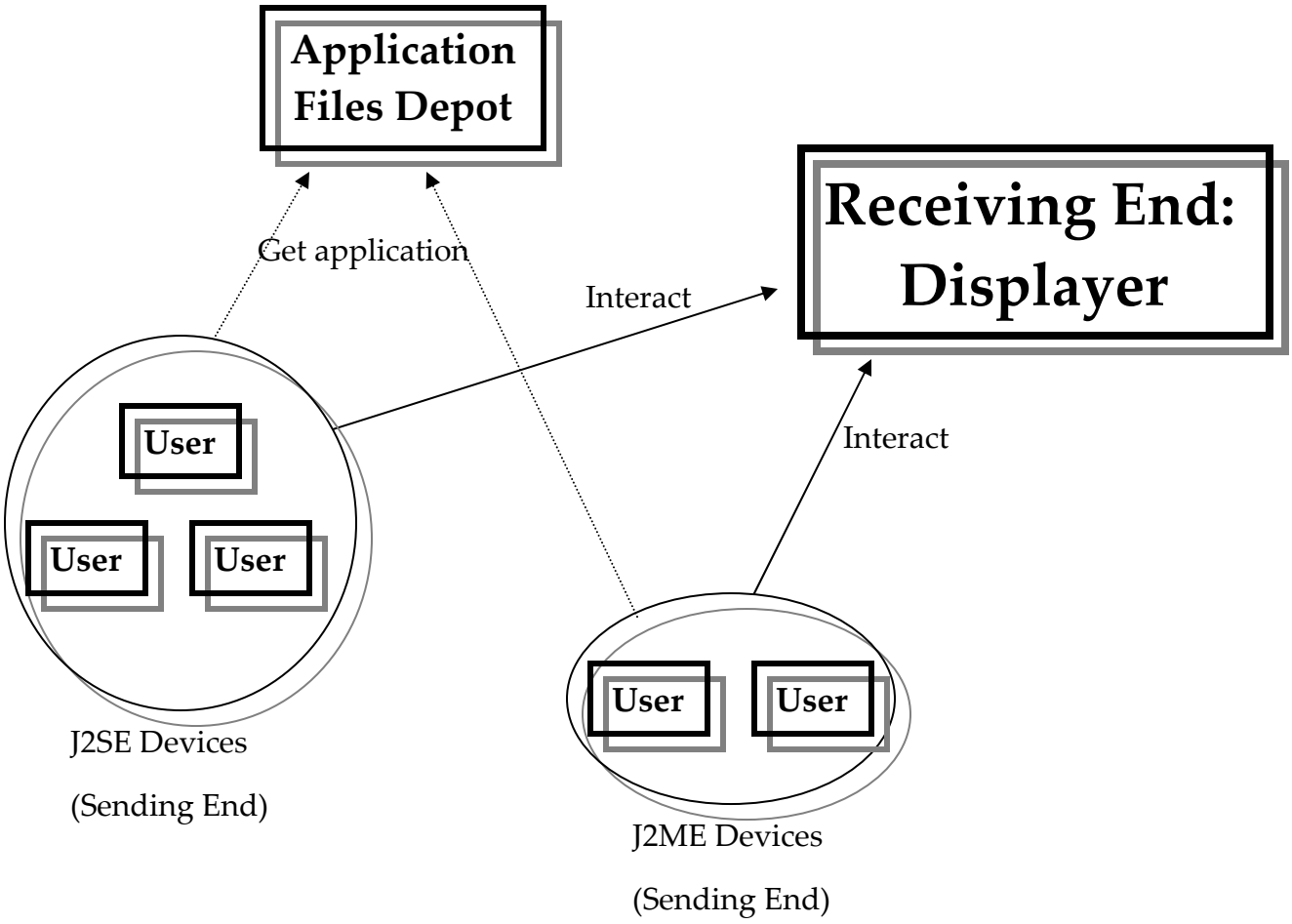
This would allow the users of non-java enabled devices to be able to access the system via a Web interface, or, by using smaller devices, via a WML interface. Unfortunately, this feature was never concluded, due to time restrictions and wasn't implemented in the final release.

8th Requirement

An important part of the system, given its uplifting nature, is the use of Graphical User Interfaces (GUIs), to enhance the user experience.

From specification to design: what does it look like?

So far, the requirements of the system evolved have been rectified and important parts have been outlined strongly enough to carry the project forward into the design phase. A rough sketch of the system to this point can be used to visualize the achievements:



This first attempt at prototyping the system will be very helpful in the next stage, as it already summarizes the progress into a simple drawing while outlining potential components of the system.

The list of user and system requirements is certainly not exhaustive and by all means flawed under the eye of professional scrutiny. However, its goal is to state the aspiration of the system as could exist within the real time frame granted for its development.

Overall Design

From specification to design: primary design decisions

The phase of requirements elicitation for the system helps draw accurate decisions concerning its final design. As a rule of thumb, software design should never add unnecessary complexity to a system. As the idea behind Rostrum is rather easy to grasp, all efforts were made into producing an equally easy design solution, in terms of comprehension, predictability and usability.

The crude sketch that concludes the specification phase outlines a vital assumption that has to be made. The network connectivity required to support the operation of the system in conjunction with the vital exchange of information between a particular component of the system with many others demonstrates the need for a Client-Server architecture.
Another critical decision is the number of independent (though interconnected) components the system must have. An excellent guideline for this derives from the former phase. A simple glance at the outline is enough evidence to support the idea of four distinct components:

1. A receiving side (a Display Server)
2. A sending side (for Clients using J2SE)
3. Another sending side (for Clients using J2ME)
4. An application files depot that enables users to download them (e.g. a Web Server)

As far the system design is concerned, this primary division into four components serves all the purposes outlined above, giving a clear and concise role to each one of them.

What were the alternatives?

However likely to function a design might look, there are aspects of such decisions that might be disregarded or incorrectly evaluated. The four-way split of the system still seems like a wise decision after a brief tour of the alternatives.

A two-way system is the fundamental part of this system, requiring only the receiving-(Server) and sending-(Client) parts to function. However, there is no compatibility between J2ME and J2SE, so either the smaller devices or the bigger ones would have had to be excluded, since there could only be one version of the Client side. In addition, the necessary feature of distributing the application files would have had to be delegated to the Server side, adding to its complexity.

A three-way system would either have to exclude a substantial part of the users, with either a J2ME or a J2SE device, or add the file-distributing feature to the Server. Either way, as explained above, the design is less than ideal.
A third option would be a multi-parted system. However, no complex logical entities exist in the four-way division that would benefit from a further partition. The most complex entity, as displayed further on, is the Server. Its features are essentially broken down, in the course of implementing them, nevertheless no logical entity can stand out; most of these, lesser, entities are incorporated in the Server itself.

What is next?

Establishing the necessity of dividing the system into four fundamental components, a discussion follows into the individual design, implementation and testing of each. Each of the four following chapters is about one of the fundamental components, i.e. the Display Server, the J2ME Client, the J2SE Client and the Web Server.

Sending end: the J2SE Client

What purpose does it serve?

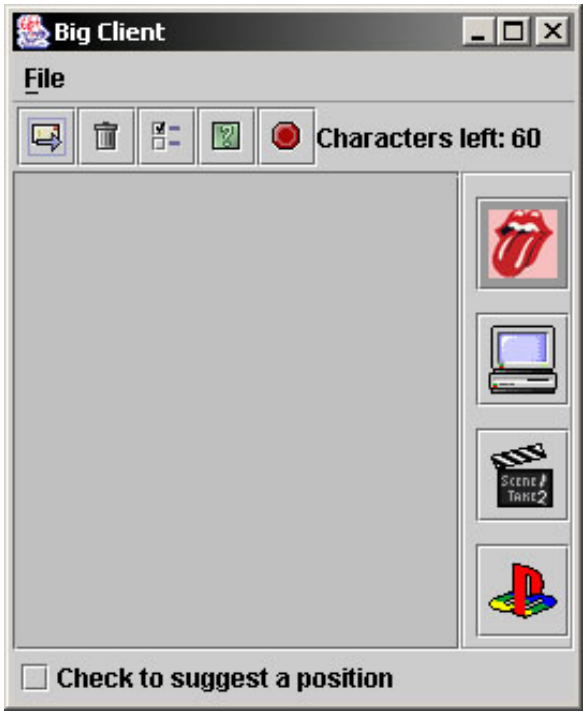
The J2SE Rostrum Client (which will be referred to as "Client" from now on) serves as the sending part for users that have the Java 1.4.1 SDK installed on their system. It is the tool that allows users of devices such as personal computers or laptop computers, either wired to a network or wireless, to access the system, i.e. to send ("post") a message.

How was it designed?

A large majority of Internet users have witnessed the evolution and consequent adoption of instant messaging systems. A relatively old idea, based on the ageing Internet Relay Chat (IRC), brought to a new level of individual gratification, most contemporary users have had an experience with some of the most famous instant messaging systems such as ICQ, MSN Messenger, Yahoo! Messenger, Trillian etc.

The design of all of the above follows a basic pattern of a small sized Graphical User Interface (referred to as GUI from now on) that contains most options needed for communication on the main screen. The Client was designed to share the same advantages of size and usability. The GUI was built for providing users with immediate access to all functions of the Client.

Implementing the Client: what does it look like?



The GUI of the Client is irreversibly entwined with its functionality. All functions and options a user might choose to use or alter are displayed as buttons with icons on the main window. This is what the application looks like when it is loaded. The original size of the window is 270x330 pixels, small-sized and discrete. The goal of the Client is to provide an interface that is easy to understand and can be used by both novice and advanced users immediately.
Some of the features described are introduced here for the first time and possibly concern or affect the receiving end of the Client's transmissions, the Display Server. In this case, these features will be taken for granted here as they will be thoroughly examined later on.

Understanding the Client functions

The Client exists for the purpose of getting a short text message across a network to the receiving server, the Display Server, where it will be displayed. This introduces the fundamental building block of this system: the Message object.

•The Message Object

The Message class creates Message objects, designed to hold whatever information needs to passed on. Some of it is supplied by the user while some is gathered by the Client itself. The information encapsulated in a Message object is:

i. the Message content

ii. the date

iii. the name of the user (username)

iv. the topic (one of the four default)

v. the user's gender

vi. coordinates x and y

The Message class creates *serializable* objects, which is necessary for the transmission mechanism used (*Serialization*).

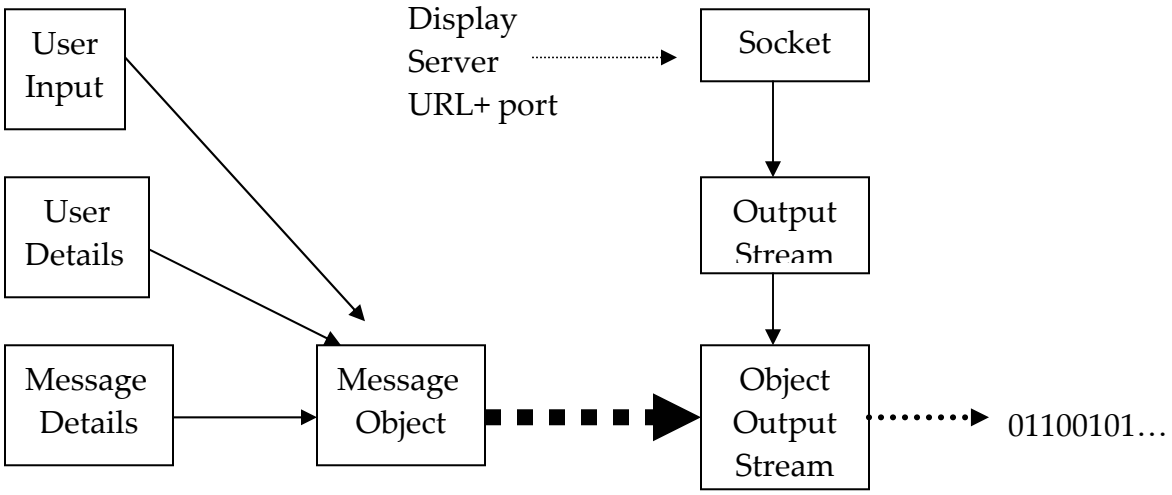
What is Serialization?
⇒ Object serialization saves an object's state in a sequence of bytes so that the object can be reconstituted from those bytes at a later time.

Java I/O, E. R. Harold, O'Reilly, 1999

The methods of the class are simple *get-* and *set-* methods that allow setting and retrieving the values above.

•The network connection

The Client uses object serialization to transmit the data necessary to the receiving end. During serialization, an object's state along with its current values is saved in a form that can be transmitted over a network connection. In its saved form, the Message object life is as follows (read from left to right):

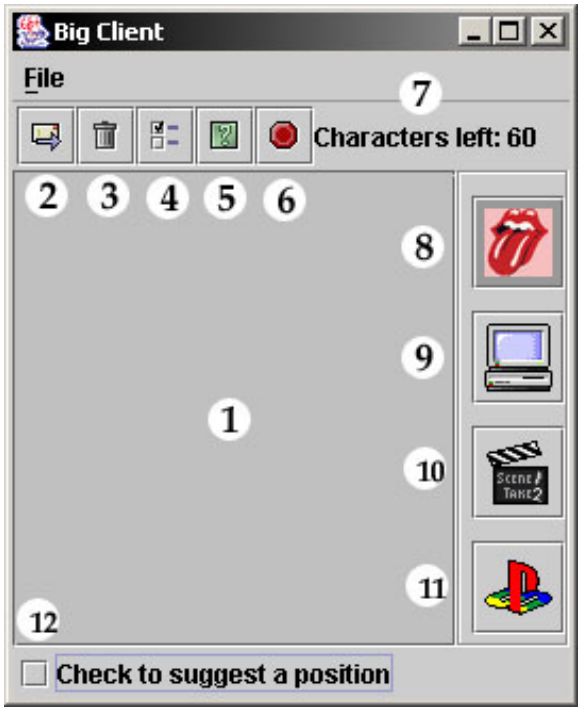


A Message object is constructed from the details gathered by the Client; an Object Output Stream (OOS) is obtained from an Output Stream which is in turn obtained from a Socket created with the Display Server's address and port. Using the *writeObject()* method of the OOS, the Message object is now a stream of bytes traveling down the network medium (whether a wire or "ether") towards the Display Server where the inverse procedure is waiting to happen.

Components of the Client

Apart from the Message class, Client is constituted by another three classes, an HTML file and several icons and pictures in .jpg and .gif format. Those classes are used to represent two important windows of Client, the configuration and help dialogs.

A small overview of the functions provided by each window of Client will give its full functionality. Numbered bullets help recognize several components.

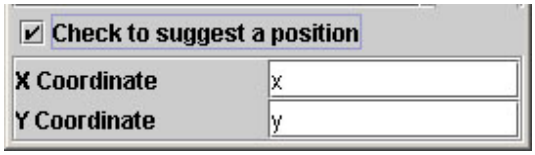


1. The main text input area. This is where a user writes her Message. Input is limited and enforced to a maximum of 60 characters. If text larger than 60 characters is "pasted" in the field, it will be truncated. When the user reaches the end of her text limit, a distinctive sound is heard when trying to input more.

2-6. The main button bar. The functions are, "Send", "Clear", "Configure", "Help" and "Exit", respectively.

7. A real time counter of characters left in the message.

8-11. The list of "Topics" supported by the Display Server. A user chooses her destination "Topic" by selecting an appropriate button. Available choices are "General Chat", "Computer Chat", "Cinema Chat" and "Games Chat", respectively.



12. This options enables users to suggest a set of coordinates that they wish to place their message to. This is a mode of posting messages best explained in the Displayer Server chapter. For now, this option should represent a set of positive x and y coordinates that are sent encapsulated in the Message object. The range (positive numbers), existence and format (e.g. non-character input) of the coordinates is enforced by the Client. Input is also limited and enforced to 2 characters. Under the terms they are used, they are more than enough.

The Configuration appears either by clicking the "Configure" button or by selecting the menu option of the right-click menu (to be shown). There are four configuration options available.

1. The Server address field is the only real prerequisite to enable communication with the Server ("posting"). The Client cannot enforce its correctness, therefore it is up to the user to input the address correctly.
2. The username (that will appear in the "post") is by default "anon" and its length is limited to (and enforced) eight characters, due to the limited display area.
3. The Web Server address was meant to be used by a feature that did not find its way into the last version of the Client. It does not play any role now.
4. The two figures represent the Male/Female option that a user can choose. The selection of gender might affect the appearance of the message. The default selection is "Male".

The right click menu appears over the writing area of the Client. It has four options, all representing existing functions already appearing in buttons, as explained earlier.

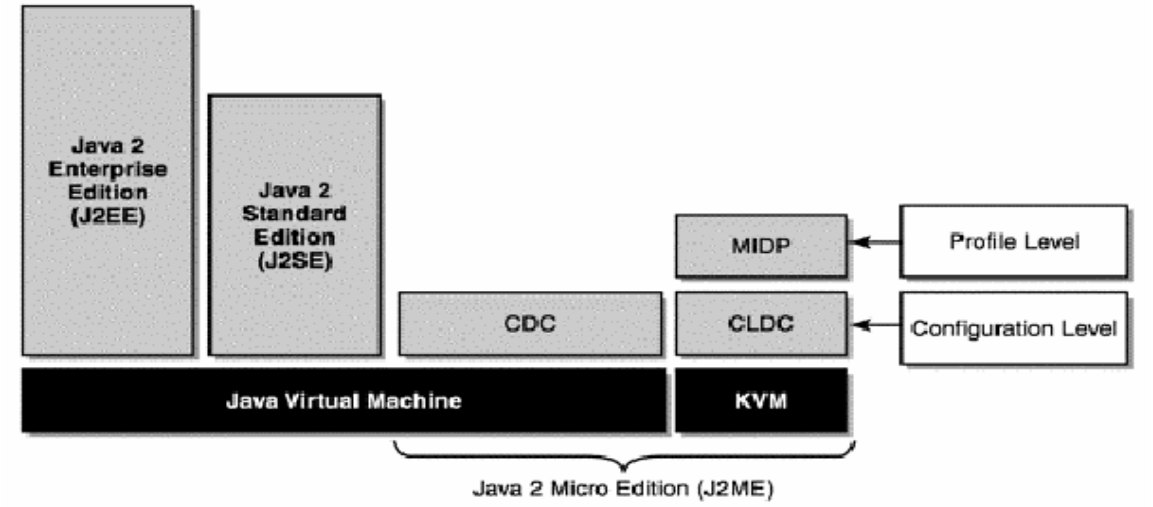
MicroRostrum: the J2ME Client

What purpose does it serve?

The J2ME Client or "MicroRostrum" is the version of the Client application developed for platforms such as Mobile Phones and Personal Digital Assistants. It uses the "Micro" edition of Java, a relatively new branch of the Java language family that will require further investigation to properly understand the J2ME Client whereabouts in the context of the system.

A necessary overview of the Java 2 Micro Edition (J2ME)

Java 2 Micro Edition is a platform developed by Sun Microsystems aimed at devices with little processing power, small display area and limited memory. As the capabilities of these devices are strictly limited, the requirements for support of the Standard edition of Java (J2SE) could not possibly be met. Thus, J2ME is by default a subset of the J2SE platform, surrendering capabilities for the sake of "weightlessness".
A schematic representation of the correlation between the existing platforms of Java will help enlighten the position of J2ME.



(J. W. Muchow, Core J2ME™ Technology & MIDP, 2001, Prentice Hall)

The devices targeted by J2ME are very diverse; they are also dissimilar in terms of capabilities. In order to address these major differences in memory, processing power and display, Sun introduced the idea of *Configurations* and *Profiles*.

What is a Configuration?

⇒ A ***Configuration*** defines a Java platform for a broad range of devices. A Configuration is closely tied to a Java Virtual Machine (JVM). In fact, a Configuration defines the Java language features and the core Java libraries of the JVM for that particular Configuration.

J. W. Muchow, Core J2ME™ Technology & MIDP, 2001, Prentice Hall

Two sets of *Configurations* exist in the J2ME world:

•CDC (Connected Device Configuration)

The high end of configurations, expects memory from a minimum 512Kb and network connectivity and lightweight user interface support.

•CLDC (Connected, Limited Device Configuration)

For lower end devices such as Mobile Phones, with lesser capabilities; 128Kb of memory are required with as little as 32Kb for the runtime environment. Display area is as small as 96x128 pixels and network connectivity is usually low bandwidth. In order to accommodate a Java Virtual Machine (JVM) in such a restricted environment, Sun developed a lighter version of the JVM, known as the K Virtual Machine or KVM.

To relieve the inflexibility and lack of features of *Configurations*, Sun introduced a set of libraries for particular types of devices, called *Profiles*, built on top of *Configurations*, extending them. The Mobile Information Device Profile (MIDP) is most fitted for all mobile devices.

What is a Profile?

⇒ …the Mobile Information Device Profile (MIDP) defines APIs for user interface components, input and event handling, persistent storage, networking and timers, taking into consideration the screen and memory limitations of mobile devices.

J. W. Muchow, Core J2ME™ Technology & MIDP, 2001, Prentice Hall

How was it designed?

A successful design of MicroRostrum would include, as depicted in the requirements, an equally satisfactory level of services provided as the J2SE version in conjunction with the usual aims of usability and lightness.

However, several negative aspects of the J2ME development environment have influenced the design and implementation processes.

The K Virtual Machine is theoretically built by Sun, as a reference Virtual Machine that is suitable for running the J2ME applications. The actual implementation of the Machine is entirely up to the manufacturer of the device. This inevitably leads to minor differences in the different KVM implementations, as manufacturers might try to optimize the KVM for their particular devices.

A fair example is the use of Java Sockets on an emulated level of J2ME. Using Sun's Studio One Mobile Phone Emulator, Socket connections were mostly stable and predictable; however, using an exact emulator of the popular Nokia 6310i Mobile Phone, sockets were not working.

Given this diversity and the fact that a single application had to be developed to fit it, the common denominator had to be chosen. In versions of *Profiles* and *Configurations*, the selected were CLDC 1.0 and MIDP 1.0, currently supported by all Java-enabled devices such as Mobile Phones and PDAs. MicroRostrum is developed and tested in both an emulated environment and a real one.

Implementing MicroRostrum: what does it look like?

The selected emulator is Nokia's Emulator for the 6310i phone series, picked to match an actual 6310i phone, where differences could be easily spotted. During the course of testing (as explained later on), behavior of MicroRostrum on both levels (emulated and actual) was always identical.

This is the initial screen of MicroRostrum, on the Nokia 6310i emulator. The user is presented with three options, to "Configure", "Post a Message" or "Help" (barely visible here). The idea behind MicroRostrum is to provide the same services as its J2SE counterpart, as explained earlier. This phone represents the minimum configuration a device is expected to possess in terms of screen size, color and processing power. The application files in a J2ME application using the MIDP profile ("MIDlet") must be archived in a single compressed file (Java JAR file) to be used by the phone. Installation can either be manual or using Over-The-Air (OTA) user - initiated provisioning. More about OTA will be explained in the Web Server chapter.
MicroRostrum is only 8Kb in total size, which satisfies the need for a lightweight application that would not encumber a small device.

Understanding MicroRostrum: how does it function?

A fundamental difference in the internal functionality of MicroRostrum from the J2SE Client is in the way the Message from a user reaches the Displayer Server. The aforementioned faulty implementation of Sockets and the lack of a serialization mechanism in J2ME, as opposed to J2SE, introduced an extra layer of difficulty, since the techniques used in Client could not be applied here.

•The solution to the network connection problem

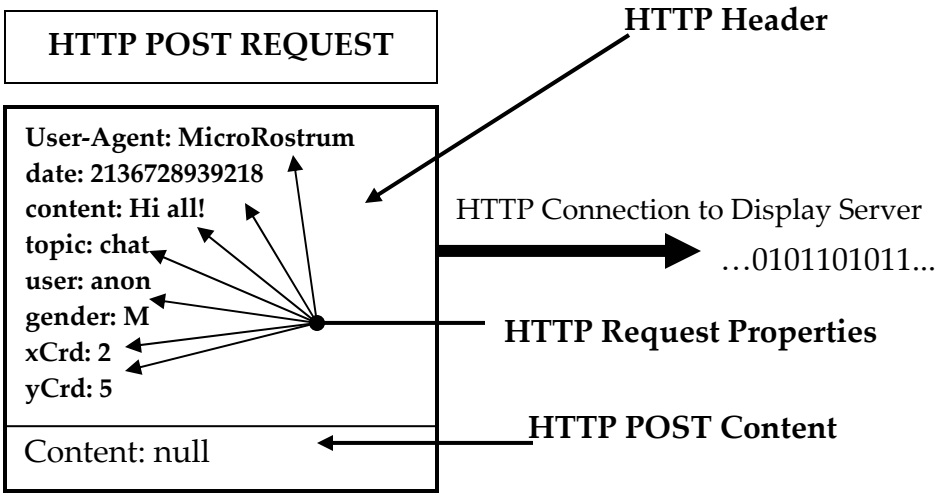
Since the availability of Sockets is limited, the only viable solution had to be found using the single most supported protocol, HTTP. MIDP version 1.0 has support for HTTP 1.1 which is a strict prerequisite to any mobile device supporting Java. Some of the alternatives that were examined included:

→ **Custom Serialization**: It is possible to create your own serialization mechanism. It would include creation of a Java Interface and implementation of a *serialize()* and a *deserialize()* methods. It would introduce unnecessary complexity, while being error prone and difficult to handle.

→ **XML data and SOAP (Simple Object Access Protocol):** A viable solution, but more suitable for large scale systems, with more complex information. It would also require the development of a SOAP Server in the receiving end.

The solution to this problem was ultimately based on the simplicity of the data that needed to be carried. The Message object that is passed on by the Client is essentially a vessel for simple text. HTTP can easily handle simple text, if it is being transported as a Request Property of the HTTP protocol.

All information that MicroRostrum wants to send is encapsulated in the header of an HTTP POST Request made to the Display Server. A convenient *setRequestProperty()* method of the HttpConnection class sets the values of these properties. The information (i.e. user message and details) is then sent like a normal HTTP POST Request will null content that will be received, processed and displayed. This is how an actual MicroRostrum request looks like:



On the three following screenshots, from left to right, is how the "Help" form is visualized (first picture) and the top and bottom parts of the "Configure" form, second and third, respectively. The options here are the same as those of the J2SE Client, implemented using MIDP's minimal GUI capabilities.
The last two screenshots show the "Post" form, top and bottom part. The message field and topic selections (only "Chat" is visible) serve the same purposes, as described above. Coordinates can be inputted alternatively, by scrolling further down the form.

Emulator
File Tools Help
NOKIA
r((o))strum Help
Welcome to r((o))strum. Please read the instructions on how to use
Back
Server URL
http://
Pick a username
[anon]
Options
Select your gender:
Male
Female
Options
Post a message
Type here..
Choose a topic:
Chat
Options
Back
Or choose where you want to place it
X coordinate:
-1
Options
Back

Rostrum: the Display Server

What purpose does it serve?

The Display Server is the exclusive receiving part of the system. Its job description, as outlined earlier, is to receive and display incoming messages. This dual activity has an impact on the complexity of its design, making it the single most complex component of the system.

How was it designed?

The Display Server has to reflect on a number of issues concerning its operation. For instance, provisions must be made to enable the application to listen to the network for incoming "post" requests. A facility must exist to enable the application to keep track all the aspects of the system and another to provide the care for the visualization part of the incoming messages.

Thus, a three-tier system is provided, where the three layers of functionality are:
→ Network Listeners

→ Accounting and Administration

→ Displaying Elements

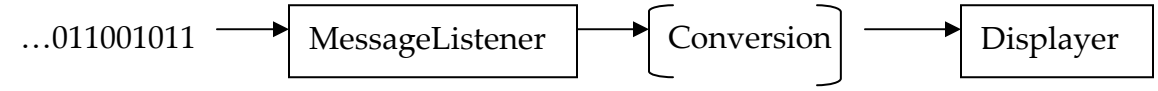
The three tiers work independently but are interlinked via a GUI. This is necessary to allow manual control over some features of the system, as will be explained later on.

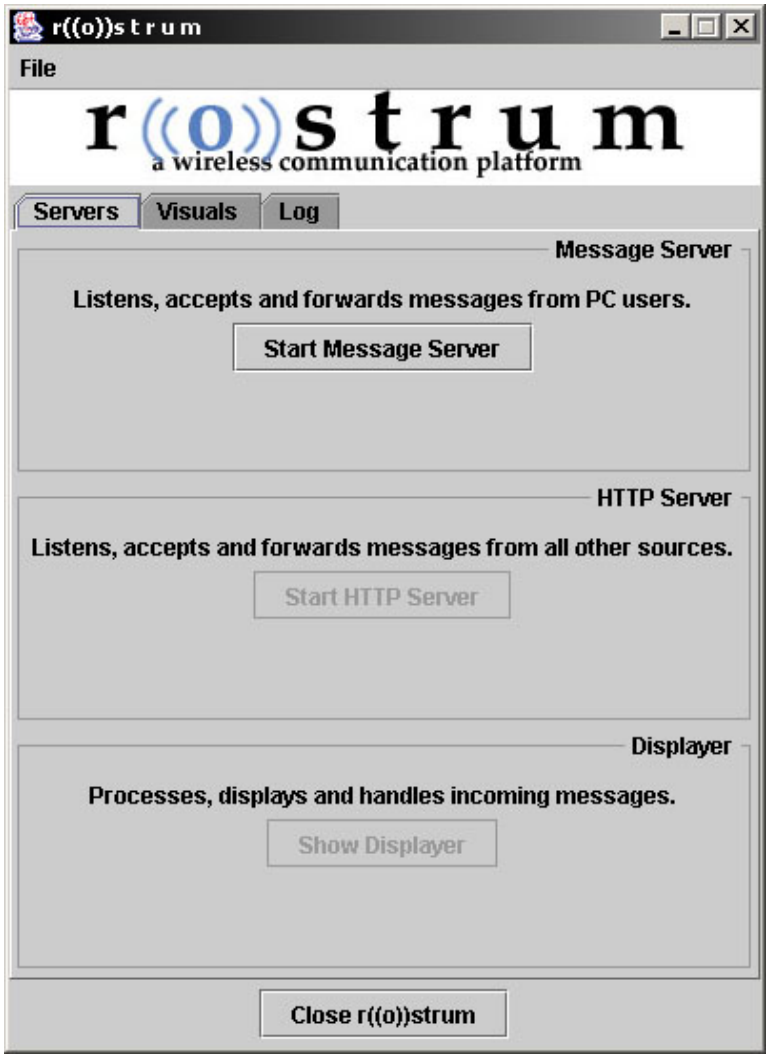
Rostrum: the Network Listeners

The network listeners provide Rostrum's ear to the outside world. The unfortunate incompatibility of the incoming messages format, from J2SE Clients and MicroRostrum Clients, led to the development of two distinct listeners. Thus, a MessageListener is created to handle incoming messages from J2SE Clients and an HTTPListener to provide for the MicroRostrum messages.

•The MessageListener

Both listeners are implemented as different threads of execution to allow constant incoming traffic monitoring. The MessageListener class is launched as a thread by another class, the MessageServer. As mentioned earlier, the J2SE Client transmits a serialized Message object, as a stream of bytes. The job of the MessageListener is to capture this stream of bytes and convert it back to a Message object. The saved state of the Message, containing all the information necessary is passed on to another very important class, the Displayer, through its postMessage() method.





This is the GUI when first launched (class RosrtumMain). There are three buttons, to start the Message server, the HTTP Server and the Displayer, respectively. It is the first of the three tabs that make up the GUI of Rostrum, called "Servers". The buttons are disabled by default and become enabled consecutively, upon activation of the previous Server.

•The HTTPListener

The HTTPListener works practically the same way with MessageListener, only at a single end. The idea behind both listeners is that, whatever kind of data they receive, it is their job to convert it to a legal Message object and pass it on to Displayer.
The HTTPListener works again, as a thread, launched by another class, the HTTPServer. It listens for incoming HTTP POST requests from MicroRostrum. When such a requests arrives, it parses the header of the HTTP request, retrieving all the information that is stored there (e.g. username, topic, coordinates etc). It then constructs a Message object and passes it on. Upon successful reception, it sends a legal acknowledgment back to the expecting device, because HTTP is a request-and-reply protocol (the default HTTP response "HTTP/1.0 200 OK", as expected by the sending device).

Accounting and Administration: Program initialization

Although the Displayer is the driving force of Rostrum, its steering is undoubtedly the job of MessageMap. The main activity of the Displayer (but not its most important) is to accept Message objects from the two listeners and display them on screen. To do this, the Displayer enforces a set of rules that apply to message posting, with the help of MessageMap. Visually, the Displayer is implemented as a full screen, non decorated window with the default Java color, grey. The GUI that controls options of the Servers and the Displayer is examined later on.

When the Displayer is launched, its primary care is to prepare the screen for message display. It does that by constructing and fitting a virtual grid to the screen, according to its current dimensions. The screen is divided into rows and columns, according to the size of the messages (by default 120x120 pixels), thus creating cells big enough for a future message to be displayed.

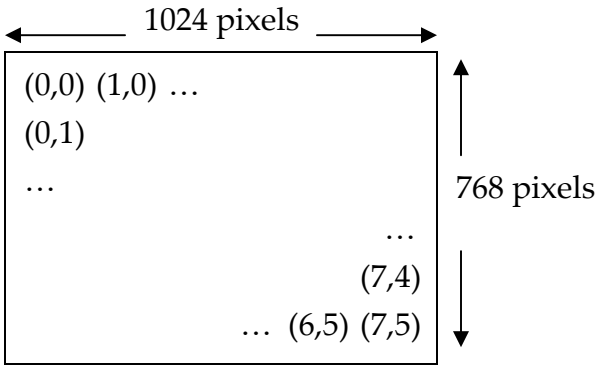
For example, given screen dimensions of 1024x768 pixels (a typical dimension):

- $\frac{1024}{120} = 8.5\overline{3} \approx 8$ (screen width over message width gives columns)

- $\frac{768}{120} = 6.4 \approx 6$ (screen height over messages height gives rows)

So eight columns and six rows are created, giving enough space for 48 messages. A screen dimension of 1280x1024 would therefore provide an 8x10 grid, with support for up to 80 messages, according to these rules.

A simple coordinate system is created to guide cells selection, interaction and manual posting by the user. The coordinates are created according to the Java rules, rather than the usual mathematical; the top left cell has coordinates (0,0) and the bottom right (in the 1024x768 case) has (7,5).



However, this partition is not enough because, as mentioned earlier, care must be taken by Rostrum to accommodate different areas on the screen, where different "topics" can be held for discussion. This leads to a further partition of the screen into four quadrants, where each topic can be accommodated. Each quadrant is assigned a topic name, as shown below, that can be used by the users to guide their messages to the topic of their interest.

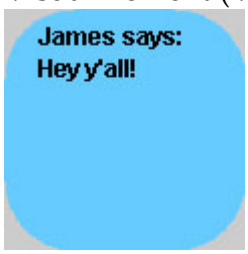
1st quadrant "Chat"	2nd quadrant "Computers"
3rd quadrant "Movies"	4th quadrant "Games"

Quadrants are handled internally as a subset of the coordinates that are created during startup. Quadrants try to cut the screen into four equal parts, so a 8x6 grid, as in 1024x768, would provide four quadrants, each of size 4x3.

Screen and quadrant partitioning are done automatically by the Displayer and MessageMap at program startup. One decision that had to be made during design of Rostrum was about Message size (screen area assigned to display it). Given that the grid is calculated according to message size and the program should work in all screen dimensions, message size was decided to be a constant, hard coded into the system because arbitrary behavior would otherwise be experienced. A huge message size in a small screen would produce unwanted results.

What is a VisualElement?

The visualization of a Message comes in the form of another object, a VisualElement (VE). A VE is the main display element of the system. Its job is to visualize a Message (here from user "James", Male). It looks rounded, based on a simple optical illusion: the four corners are individually drawn with the background color, which, in the case of the Displayer, is grey. It is, as mentioned, 120x120 pixels in size and its attributes are same as those of the Message it encapsulates, in addition to color and position on the screen.



•Color in VisualElements

These are two color modes that Rostrum uses; gender-based colors and quadrant-based colors. Gender-based color mode paints the VEs blue if the user has selected Male in his Client configuration or pink if she has selected Female. Quadrant-based coloring uses a different color for each of the quadrants: orange, red, green and light purple for each of the four quadrants, respectively.

•Position and Information

If you right click on a VE, you can see a dialog with its individual information. On the right is seen what this dialog looks like, with information about positioning, user, topic, gender, date and time of the post. The coordinates given correspond to the relative position of the VE in the virtual grid of the screen. The dialog is represented by the VisualElementInfo class.

How does it all work together?

•The Displayer

The Displayer is a class that has facilities to

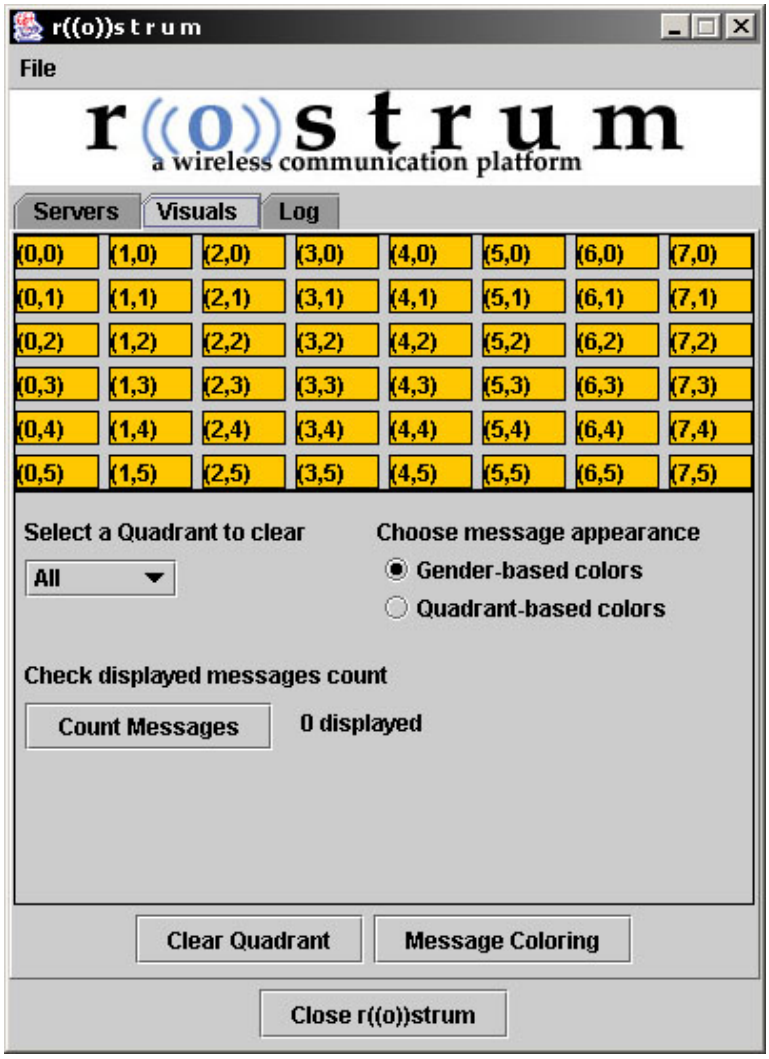
→ post Messages

→ remove Messages

→ clear quadrants of Messages

→ count Messages

→ set color mode (gender- or quadrant-based)



All of the above options can be controlled via the second tab of the GUI, named "Visuals" (class VisualsController). The upper part of the tab visualizes the virtual grid as it was formed by the Displayer. It changes dynamically to fit the current grid of the screen. Here, a screen dimension of 1024x768 was used at program startup and the grid created was 8x6 as explained previously.

Below the grid are the controls that enable the alternative configurations of the Displayer, such color mode, quadrant clearance and message count. In the case of the removal of an individual VE, a right-click menu option on any of the orange colored cells of the grid (class VisualThumbnail) enables the moderator to proceed to message removal.

In order to post a message, the Displayer has to create a VE (visualized encapsulated Message) and position it on the screen. The positioning is done using a *point* on the screen, where the top leftmost pixel of the VE will be. The point is given as a set of x and y values in pixels, starting from the top leftmost area of the screen as (x, y) = (0, 0). The virtual grid allows great control over the positioning of the elements (as to not overlap, for example), but there must be a

way to convert relative positioning information, such as the coordinates used to an actual position on the screen.

Displayer uses a simple formula to allow this conversion:

x in pixels = (relative x)*(total number of columns + message height)

and

y in pixels = (relative y)*(total number of rows+ message width)

where relative x and y are in the form of e.g. (x, y) = (2, 2) of the virtual grid.
For example, a Message that needs to be posted at relative position (2, 3) on a 1024x768 screen with a sized 120x120 message will be positioned at:
x = 2 * (8 + 120) = 256

y = 3 * (6 + 120) = 378

The top leftmost pixel of the VisualElement will be at (256, 378) on the screen. The addition of the number of columns (rows) to the message height (width) is not absolutely necessary but it allows VEs not to be too close together.

•The MessageMap

The MessageMap is Displayer's right hand assistant. While Displayer is the class that actually positions the VEs physically on the screen, it still needs a relative position (a set of relative coordinates x and y) to do so. Thus, the job description of the MessageMap includes the following:

→ providing a facility to store and retrieve the references to the VisualElements that appear on the screen.

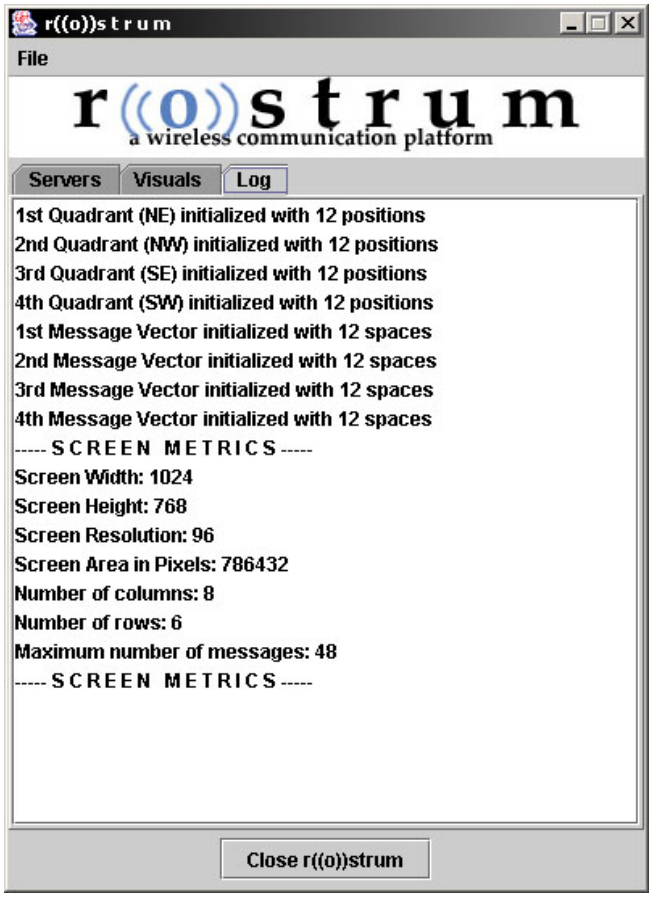
→ keeping track of what VE is positioned where

→ supplying the Displayer with coordinates of a random empty spot of screen when requested

→ initializing the four quadrants with their coordinates, according to the virtual grid calculated by the Displayer

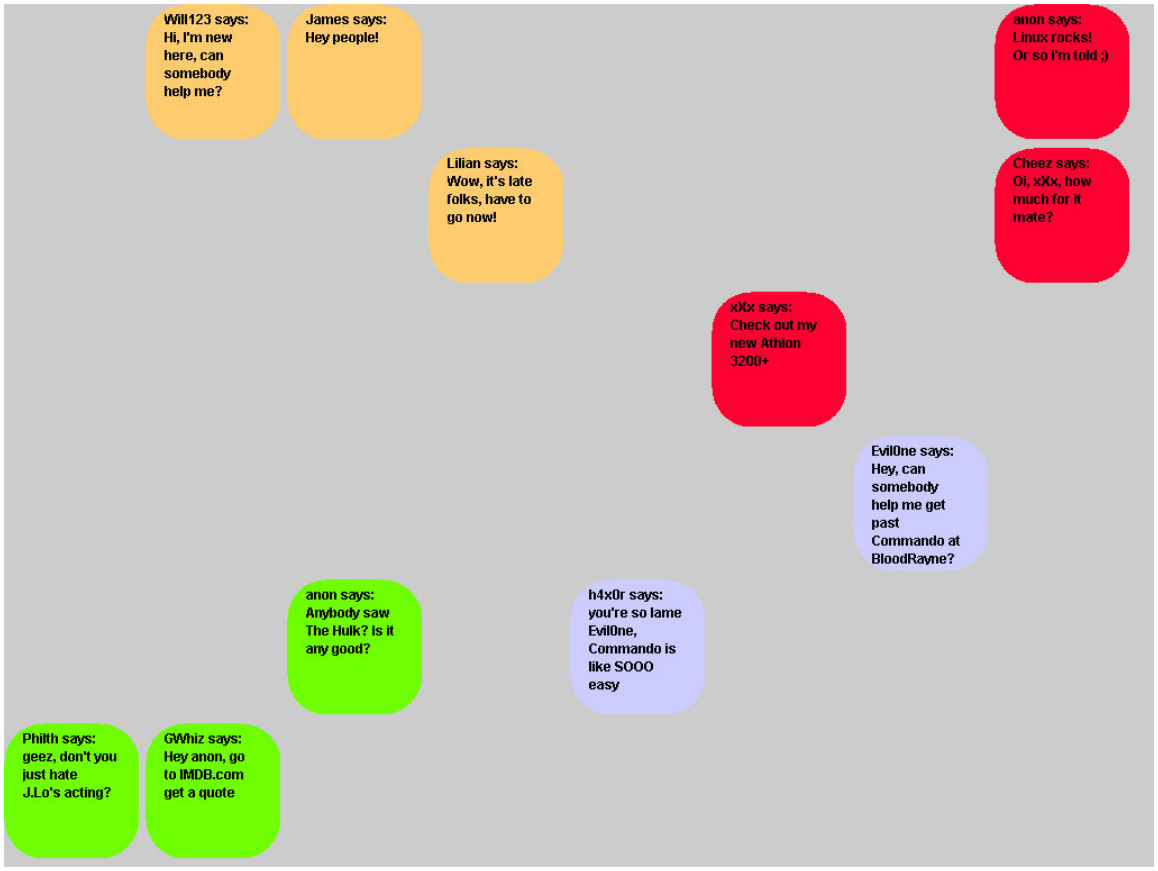
The MessageMap needs its own initialization phase; it is created by the Displayer given the dimensions of the screen grid that it uses to calculate the four quadrants. In order to communicate seamlessly with the Displayer, a Coordinates filetype is created that is simply a storage of two integers, x and y. The Displayer, in the process of posting a message, asks the MessageMap to supply a position on the virtual grid, in the form of Coordinates.

In order to provide a random set of Coordinates every time the Displayer asks for the next position, MessageMap keeps four "pools" of every available set of Coordinates (empty positions on the screen). There are four Vectors (Java filetypes) that MessageMap assigns to each quadrant and fills them with the Coordinates that correspond to their position on the screen (at the initialization stage). When Displayer asks for the next empty position on a certain quadrant, MessageMap picks randomly a set of Coordinates from the relevant Vector and removes it from the "pool". The Coordinates are received by the Displayer, a VisualElement is created with an encapsulated Message and is assigned these Coordinates. Afterwards the VE is passed back to the MessageMap to store its reference.
The reverse procedure occurs when a VisualElement is removed: the Displayer orders the MessageMap to remove it from its lists of displayed messages and the Coordinates that it occupied are reinserted in the Coordinates "pool". Then an order is given to the VE itself to dispose itself.


r((o))strum
File
r((o))strum
a wireless communication platform
Servers
Visuals
Log
1st Quadrant (NE) initialized with 12 positions
2nd Quadrant (NW) initialized with 12 positions
3rd Quadrant (SE) initialized with 12 positions
4th Quadrant (SW) initialized with 12 positions
1st Message Vector initialized with 12 spaces
2nd Message Vector initialized with 12 spaces
3rd Message Vector initialized with 12 spaces
4th Message Vector initialized with 12 spaces
----- SCREEN METRICS -----
Screen Width: 1024
Screen Height: 768
Screen Resolution: 96
Screen Area in Pixels: 786432
Number of columns: 8
Number of rows: 6
Maximum number of messages: 48
----- SCREEN METRICS -----
Close r((o))strum

The last tab in the GUI is the "Log". Here, certain classes of Rostrum can add some comments or error messages of their operation. This is what the "Log" looks like after the program is run, at a 1024x768 screen, prior to launching the network listeners. The initialization phase of MessageMap and Displayer is visible via the messages.

A last interesting point of MessageMap is the continuity it provides to message posting. The Vectors that are used as references storage for VEs are also used as a queue that guarantees sufficient posting space: if a quadrant starts filling up, MessageMap removes the 4 oldest VEs (front of queue) and trims the queue. New VEs are always added at the back.
Finally, MessageMap assists Displayer into counting the displayed VEs and clearing quadrants.
Here is how the Rostrum looks with a couple of Messages in each quadrant, using quadrant-based colors.


Will123 says:
Hi, I'm new here, can somebody help me?
James says:
Hey people!
anon says:
Linux rocks! Or so I'm told ;)
Lilian says:
Wow, it's late folks, have to go now!
Cheez says:
Oi, xXx, how much for it mate?
xXx says:
Check out my new Athlon 3200+
EvilOne says:
Hey, can somebody help me get past Commando at BloodRayne?
anon says:
Anybody saw The Hulk? Is it any good?
h4x0r says:
you're so lame EvilOne, Commando is like SOOO easy
Philth says:
geez, don't you just hate J.Lo's acting?
GWhiz says:
Hey anon, go to IMDB.com get a quote

A small right click menu is available on the grey background of the Displayer (showing on the left). Its options include "Show Config", "Suspend Displayer" and "Exit". The latter, understandably, exits the program. The first brings the GUI of Rostrum back in focus while the second temporarily suspends the Displayer and all the VEs, showing the desktop of the operating system. Displayer can be reinstated by pressing its button on the GUI.

Program Files Depot: the Web Site

The need for a Web Site

The need for a Web Site is supported by the fact that it is the easiest and most prevalent way of deploying information and file download capabilities on a network or the Internet. An out-of-the-box version of an open source Web Server such as the world favorite Apache Web Server is enough to handle all the needs of the system.
A Web Site is used by J2SE enabled users to download the application files and run them on their local system. In the case of J2ME users, the option are two: either download the application files and install them manually on their device or use a practice called Over-The-Air (OTA) user initiated provisioning.

What is OTA user initiated provisioning?

Sun Microsystems quickly realized that a MIDlet is a difficult application to deploy. Manual installation, such as mentioned above, cannot be done by a novice Mobile Phone user. Thus, a way must exist to facilitate both novice and advanced users downloading the application to their device. This is why OTA transactions came to force.
Sun published a document called "Over The Air User Initiated Provisioning Recommended Practice for the Mobile Information Device Profile" (http://java.sun.com/products/midp/OTAProvisioning-1.0.pdf) that set the guidelines for J2ME developers on how to deploy their applications.
A typical MIDP application (MIDlet), is comprised of two parts: a JAR file and a JAD file. JAR files are archives that contain all the class files and resources of a MIDlet (code, images, sounds etc.). JAD stands for Java Application Descriptor and is a simple text file that holds information about the particular MIDlet such as size, location that can be downloaded, version, author etc.

In a Mobile Phone, a Java Application Manager (JAM) resides, that is responsible for handling and running the Java applications. If we follow the OTA practice, in a normal scenario, a user is clicking on a link that links to a JAD file in the WAP

browser of his Mobile Phone. The JAD file is downloaded and processed by the JAM (e.g. checking if there is enough space in the device to download the JAR file) and should nothing goes wrong, the JAR file is downloaded and installed. The Web Site comes in force when the JAD file needs to be downloaded. However, due to firewall restrictions, OTA could not even be tested and thus was never fully implemented.
Another important job of the Web Site was to enable Mobile Phone users with non Java enabled phones to access and post messages via a Java servlet in the form of a WML (Wireless Markup Language) file. Firewall restrictions and trouble developing and testing this component caused its removal from the final version of Rostrum.

Subsequently, with several important components not implemented, the Web Site can merely serve as a software depot, an extension to the user manual and a gallery of screenshots of the system.

Testing Rostrum

The testing components

Due to the nature of the system's components' development, constant informal testing was performed throughout their lifecycle. However, formal testing of the system as a whole was conducted at the end, concentrating along these:

→ Clients' GUI and input

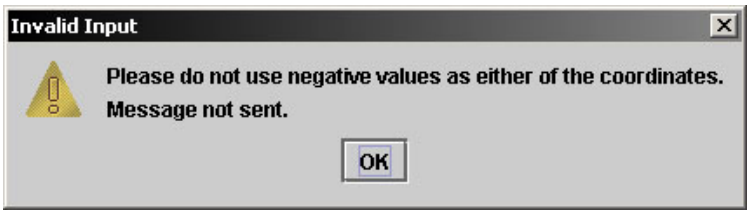
→ Clients' network connections

→ Server GUI and stability

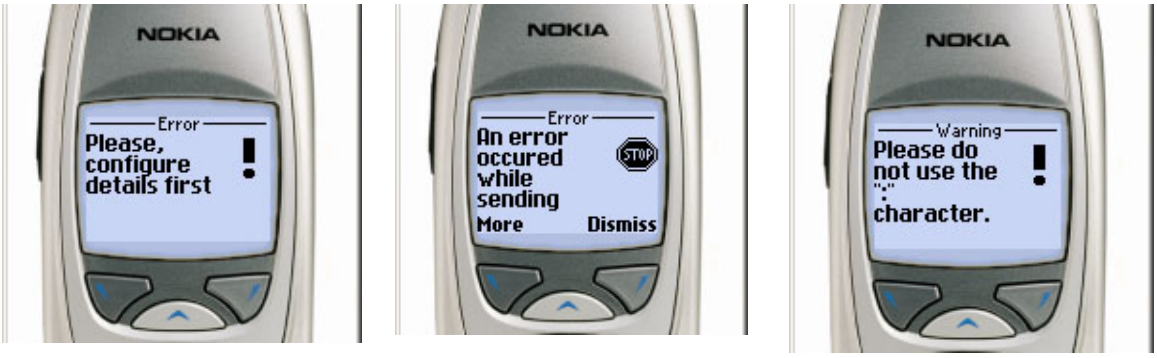
→ Variables' boundaries, initialization and value errors

Clients' Testing

Most of the errors expected to be caught in the Clients' version of the system are user input errors. The GUI of both Clients is relatively simple and is proved stable under all circumstances. Both being developed early, they received thorough testing under normal and abnormal circumstances of operation.
These are some of the comprehensive error dialogs used to alert the user of an erroneous situation:



The top left dialog serves as a reminder to users that are trying to "Send" a message without configuring her details first. As mentioned earlier, the only true prerequisite for posting a message is to input the correct address of the Server.
Nevertheless, the exactness of the Server's address cannot be checked and it is taken in good faith. If it is not correct (or a general connection error occurs) the bottom right dialog appears.
Another source of potential erroneous input is the coordinates input fields (when user tries to post her message in "coordinates" mode). If the user doesn't use numbers or uses negative numbers as coordinates, she gets the top right and bottom left error dialogs, respectively.
All input fields that need limited input, such as the main message area (limit 60 characters), username (limit 8 characters), x- and y- fields (limit 2 characters) are enforcing their limits using a custom *KeyListener* that counts every inputted key.
In the case of MicroRostrum, same basic rules apply as to user input characters limitation, with a few minor differences.



The first screenshot appears if the user tries to "Post" a message without configuring the Server details, an equivalent of the error dialog of the J2SE Client. The second screen appears when the connection is erroneous or the server address is incorrect. Coordinates "mode" input format is still checked but, as explained in the "Help" section of MicroRostrum, how her message is sent is entirely up to the user (inputting "-1" at either x- or y- coordinates field cancels coordinates "mode", while any other legal, value activates it).
The only extra restriction a MicroRostrum user is imposed is not being able to use the ":" character in his message. HTTP headers use the ":" character to separate fields; it is consequently used in the HTTPListener class too, to separate the incoming fields when the HTTP header is parsed. Due to the nature of parsing, if a user used a ":" anywhere in her message, the subsequent characters would be disregarded.
The limits of input fields are provided as options by the J2ME so no extra checking is necessary.
No unexpected erroneous situations arose during testing, apart from those of the predefined test parameters, as shown above.

Server Testing: the Displayer

The Displayer testing by itself is nothing more than test of the GUI, as it is more complex than those of the Clients. The reason for its simplicity is that Clients take a substantial load off by making sure, through their checking mechanisms, that a Message that leaves them is legal and contains no errors.
The first part of the test checks the options provided by the "Visuals" tab in the GUI, such as disposing individual messages using the grid, clearing quadrants, counting messages and changing color mode. This is essentially a test of the routines used by Displayer and MessageMap. All errors that occurred during

development had been fixed and no unexpected results were experienced (such as uninitialized variables, array boundaries errors etc.).
Only one minor "bug" was found in Displayer that could not be fixed. As mentioned before, the VisualElements use an optical illusion to look rounded, by painting the edges the same color as the background. However, if another window, dialog etc. is active on top of one, these edges seem to temporarily lose their initial color, resulting in an annoying distortion. If the Displayer is suspended and reinstated, it causes the VEs to redraw and return to their initial state.

Network testing

The final aspect of the system that needed testing was the networking components. The MessageListener responded as intended to erroneous conditions such as absence of a Message class file. Generally, both servers are instructed to shut down should a network error occurs and add a discrete message in the "Log" tab for the moderator to restart them.
Both servers (and the system as a whole) passed the "stress" tests conducted, that is being persistently bombarded with messages of both types.
A significant effect on the testing of the program was caused by the existence of two strict firewalls that filtered outside connections. No solution was found to circumvent this (it's hard to outsmart experienced engineers). The only viable solution was to use port 80 for incoming HTTP connections made by MicroRostrum, which in turn failed miserably.

Project Management

Building a significant piece of software requires both experience and expertise. No matter how good the theoretical background of how a system should be developed is, there is always a significant gap between knowing how to do something and actually accomplishing it.

The timetable for this project was outlined in the project proposal, at the beginning and was closely followed, given the essential errors, at any rate.

	1st Iteration	**2nd Iteration**	**3rd Iteration**	**4th Iteration**
J2SE + Rostrum J2ME Web Site Testing	3 Weeks	2 Weeks	2 Weeks	2 Weeks

The proposal suggested a 10 week projected time of development (on the table above are the first 9 weeks, the last was devoted to complete testing) and an extra few days break. The important part of the project was the initial three weeks period, by the end of which the prototypes were built. From there on, the development was, on the whole, according to the project's proposal.

The only differentiation from the proposal was in the last iteration, when some work on the Web Site (that was scheduled but not implemented, as mentioned) was reallocated on the other parts.

Appraisal and Conclusion

Overview

If we put Rostrum's proposal and specification side by side, we will see that most of the original thoughts about how it should be were in both parts. Implementation proved that the initial plans were, as usual, ambitious. The distance between the two is created by real world problems and errors.

On the whole, Rostrum's development can be deemed as successful as far as design versus implementation is concerned. Testing proves that Rostrum solidly realizes the features that it promises:

→ Ease of use
→ Lightweight application
→ Broad Accessibility
→ Hassle-free operation

The actual functionality of the system, as far as availability of options is concerned, proves adequately rich.

Rostrum: the Good,

Rostrum proves to be stable and efficient in terms of normal and increased operation (in terms of Messages). Messages are displayed in a vivid environment and the common veneer of the system is that of good-natured and entertaining attitude. The approach that was chosen, points towards younger audiences, in places that they meet. Anyone can easily imagine the system projected on the wall of the local pub, the common room in a School or the cafeteria.

However, the nature of the system allows it to easily extend and form a common base for every communication need that a shared space can fulfill. Instead of a

cafeteria, it can be used in a conference room to post questions to the speaker or a hospital to be used as notice board.

Nevertheless, while is the idea is significantly extensible, it had to be tailored to meet the time limitations and capabilities needed by a novice programmer working on a 10-week project. The result is fully operational in its own right and has provided this novice programmer with significant software engineering experience.

The Bad

As much as Rostrum is successful, there can be doubt that some elements could be made better, extended or even taken off the project. Time is always against anyone with a date circled in red in her calendar; to a software engineer the definition of a "deadline" is equivalent to the day one stops adding features to her work; new ideas and "bugs" emerge on a daily basis.

A few important features that were scheduled and designed never appeared in the final version of Rostrum. One of them was the WML page that would allow user of any Mobile Phone to post a message to Rostrum via her WAP browser. Engineering implications and time restrictions never allowed it to complete, therefore removing an important element that would provide universal access to the system from all mobile phones. Another feature developed and not implemented due to time limitations was the ability of the J2SE Client user to view the board on her computer. A class was designed that took screenshots of the full screen every x seconds and saved them in .JPG format but did not appear in the final version as well.

The absence of any form of security or authentication is due to the fun nature of the application. The only abuse of the services provided would be inappropriate language that could only be moderated by filtering incoming messages. However, censorship and fun generally don't appear in the same sentence.

And the Ugly

One of the major sources of errors, misunderstandings and flaws that were experienced in the course of development were the Java 2 Micro Edition language and tools. Throughout the time of development, J2ME and its various tools proved unreliable and inconsistent. The same code would behave differently on a different emulator, or not at all. Such is the nature of the J2ME language that the same code behaves and looks quite different in various available devices. The literature on J2ME is blossoming but the general consensus seems to be that the language is still in its infancy and some way off of being a mature programming platform. Sun has released the specification for MIDP 2.0 and CLDC 2.0, with some major improvements but support for them by today's devices is nearly non-existent.

Finally, an unsurpassable obstacle that had a major impact on the design and implementation of the system was the existence of firewalls. The School of Computer Science's firewall is extremely strict (as opposed to the more lenient University of Birmingham firewall) and could not be reached by any outside connection.

References

●Andrew S. Tanenbaum, **Computer Networks**, 4th Edition, 2003, Prentice Hall PTR
●John W. Muchow, **Core J2ME Technology & MIDP, 2001**. Prentice Hall PTR
●James White & David Hemphill, **Java 2 Micro Edition, Java in Small Things**, 2002, Manning Publications CO
●Elliotte Rusty Harold, **Java I/O**, 1st Edition 1999, O'Reilly

Bibliography

●Ian Sommerville, **Software Engineering**, 6th Edition, 2001, Addison-Wesley
●Elliotte Rusty Harold, **Java Network Programming**, 2nd Edition 2000, O'Reilly
●Marc Loy, Robert Eckstein, Dave Wood, James Elliott & Brian Cole, **Java Swing**, 2nd Edition, 2002, O'Reilly
●Kim Topley, **J2ME in a Nutshell**, 2002, O'Reilly
●Ian Darwin, **Java Cookbook**, 2001, O'Reilly
●Andrew S. Tanenbaum, **Computer Networks**, 4th Edition, 2003, Prentice Hall PTR
●John W. Muchow, **Core J2ME Technology & MIDP, 2001**. Prentice Hall PTR
●James White & David Hemphill, **Java 2 Micro Edition, Java in Small Things**, 2002, Manning Publications CO
●Elliotte Rusty Harold, **Java I/O**, 1st Edition 1999, O'Reilly

Appendices

Appendix A: Project Proposal and Timetable

Introduction

The evolution of wireless networking technologies has raised consumer expectations to deem powerful ubiquitous mobile computing as a standard. Concurrent advancements in mobile devices' capabilities have forged a new area of software development, aimed specifically at small and portable devices rather than the usual mass of desktop computers.

The proposed system aims at the majority of all types of small devices capable of simple networking as well as the entirety of the more complex and advanced desktop computers. These devices range from Mobile Phones to PDAs, Pocket PCs, Laptop and Desktop Computers.

The System

The proposed system will aim at providing an open shared space (e.g. a computer screen, a TV, a projection on a wall) that every wireless device can interact with. The interaction will be in the form of short messages displayed ("posted") in the shared medium, where they can be viewed by all.

These messages will be displayed as separate entities in an appealing graphical environment rather than a simple monotonous list. Messages will be displayed pseudo-randomly on the graphical environment, grouped in regions by topics. Ideally, the use of "smileys" may be implemented (user could change the appearance of her message, based on predefined behaviors).

The goal of the system is to provide a user-friendly interface where no significant computer literacy is required, transparency, playful nature, optional security features (strict security is beyond the scope of such a system) and above all a plethora of supported devices.

A negative aspect of the latter assumption is the base on which such a system would work. Due to the major differences of the targeted devices' capabilities, the lowest common denominator must be used to relieve the otherwise overwhelming complexity of device-specific context handling. However, the devices have to be split up into two major categories, for each of which a different client application will be built.

The overall objective is to deliver a message from a user to a shared medium, using any device. The ways of achieving this is by using a mixture of cutting edge technologies with applied ones. The application will be based on two flavors of Java, J2SE (Standard Edition) and J2ME (Micro Edition).

Components of the System

The common aspect of the targeted devices and also their minimum requirement to use the system will be access to the Internet. Apart from older devices, nowadays practically every device has Internet access while a subset of them supports the Micro Edition of Java (J2ME). Based on these assumptions, the components of the system are:

1. Client Applications

In accordance with their level of support for the Java language, smaller Java-enabled devices (Mobile phones, PDAs etc.) will communicate via a J2ME application (MIDlet) with the Display Server (see below) using simple HTTP protocol. For perceptible compatibility reasons, the client MIDlet will be built using CLDC 1.0 (Connected, Limited Device Configuration) and MIDP 1.0 (Mobile Information Device Profile) currently supported by nearly all target devices. A light graphical user interface with some basic options will be used for that reason.

Devices that support J2SE, such as Laptops and Desktop Computers will use a more delicate client application, with more options and a better graphical user interface. Communication will be done using Peer-To-Peer architecture with the Display Server (TCP/IP Sockets).

Non-Java devices should be able to use a limited version of the system via a Web Interface provided by the Web Server (see below) via their built-in WAP Browser.

2. Web Server

The Web Server (e.g. a simple Apache Web Server) lies in the heart of the system, acting as a mediator between the Clients and the Display Server with three major tasks.

i. Serving the information and help pages of the system (in both .html and .wml format, suitable for normal and WAP Browsers accordingly)

ii. Serving the Web Interface of the system for posting messages (in both formats as above, .html and .wml)

iii. Serving the application files for both J2ME (.jar file) and J2SE (.zip or .gz) to be downloaded by the devices

The first task is accomplished by static .html and .wml (Wireless Markup Language) pages, describing the system and providing extensive help on its features.

The Web interface is a simple form for posting messages. An underlying Java servlet will be responsible for passing the messages to the Display Server. The Web Interface will be used as a last resort substitute of the original program, designed to provide the core functionality of the system to those devices that cannot or will not install and use the client application for that purpose.

Finally, the most significant task is to provide a means for the potential users to obtain the application files. In the case of a J2ME client, Sun's Over-The-Air (OTA) User Initiated Provisioning recommended practice gives the guidelines of how to transfer a MIDlet form a server to a client. Those guidelines are currently followed by all concerned parties (device manufacturers). A small text file called .JAD file (Java Application Descriptor) that gives the target device (and user) an insight on the application specifics is first downloaded and read. Should the device (and user) decide to install it, the .JAR files (Java Archive) containing the application files is downloaded. If a problem occurs (OTA transfers are Network Operator dependant), the user can either download elsewhere and install manually into the phone (or other device) or she can use the Web Interface.

3. Display Server

The Display Server is a Java J2SE application with the task of receiving incoming messages and displaying them on the shared medium. Sources of incoming messages are:

i. An HTTP POST request from a J2ME application
ii. A request via a TCP/IP Socket from a J2SE application
iii. A request from the servlet controlling the Web Interface

Major tasks of the Display Server are:
i. Receiving and displaying incoming messages deciding where and how they will be presented
ii. Uniquely identifying devices (using cookies for J2ME devices)
iii. Enforcing the required level of security
iv. Providing an administration interface for setting server parameters
v. Logging of required actions

Project Timetable

The system will be scheduled to be built as a 10 week project. It will be built in four iterations, spread unevenly across the timetable. Each iteration will involve the following components:

1. J2SE Client development, Display Server development
2. Web Server development, Web Interface development, Web Pages building
3. J2ME Client development
4. Testing of each component

The first iteration will last three weeks due to the building of the prototypes, while the next two will last two weeks each. In the last three iterations, all components will be refined and, time permissive, new features will be added. In the remaining week, full system tests will be conducted (components and component relationships are to be tested at each iteration).

Appendix B: Running the Software

The sources and binary files of the application are under the directory "Dissertation" in my home directory /msc46nxp. There are four directories inside:

●**/micro:** contains the sources, JAR and JAD files for the J2ME application. The emulator used in the demonstration of the software is Nokia's 6310i emulator from Nokia Developer Suite for J2ME v.1.1.

●**/html:** contains the sources of the Web Site. It can run via the **index.html** file

●**/rostrum/client:** the sources of the J2SE Client. Can be run under windows using the BAT file provided. Detailed help can be found using Client's Help file

●**/rostrum/server:** the sources of the Server. Can be run under windows using the BAT file provided. Detailed help can be found in this document in the appropriate chapter.

Appendix C: A UML Sequence Diagram of how a Message is posted using Rostrum and J2SE Client